World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:4, 2007

Using Interval Trees for Approximate Indexing of Instances

Khalil el Hindi[1]

Abstract**—**This paper presents a simple and effective method for approximate indexing of instances for instance based learning. The method uses an interval tree to determine a good starting search point for the nearest neighbor. The search stops when an early stopping criterion is met. The method proved to be very effective especially when only the first nearest neighbor is required.

Keywords**—**Instance based learning, interval trees, the knn algorithm, machine learning.

I. Introduction

The knn algorithm [1] is an important form of Instance-based learning. It is a lazy learner, because it simply retains a number of classified examples and uses them to classify new examples. The instance memory, where we retain instances, is usually represented as a table. Each row represents a classified instance and each column represents an attribute (feature), with one column designated as the class attribute.

To classify a new instance, the most similar (or the nearest) k instance(s) are retrieved. The class of the example is predicted to be the class with majority votes among the classes of k instances. k is usually chosen as a small odd number such as 1, 3, or 5.

To achieve good classification accuracy, the knn method retains a large number of classified examples, which requires storage space and slows the classification process. Several methods appeared in the literature that attempt to address these two problems. These methods can be categorized into two main categories: reduction techniques and indexing techniques. Reduction techniques [7] reduce the size of the instance memory by retaining the most informative instances (examples) and eliminating irrelevant instances. An example is considered informative (relevant) if removing it has a substantial effect on the classification of other examples. Reduction techniques make a tradeoff between accuracy and storage. Furthermore, these methods are not suitable if all instances must be retained for reasons other than classifications. Indexing techniques using k-d trees [4,6] were developed to reduce the time required to find the nearest neighbor without eliminating instances. However, they are most suitable for applications with a low number of attributes. They become less effective when the number of attributes increases [7].

In this work, we investigate the use of simple interval trees, combined with early stopping criterion [2], for approximate indexing of instances. The aim is to avoid searching the whole instance space to find the k nearest neighbors. Results show that the proposed techniques is highly effective specially when k=1.

In section 2, we review the similarity function we used in our experiments. Section 3 reviews the early stopping criterion we use. In section 4 we present a simple form of range trees that we used in our experiments. Section 4 discusses our results. Section 5 discusses our main conclusions and direction, for future work.

II. Measuring Similarity between Instances

Despite their simplicity, instance based learners proved to give good classification accuracy in many applications, comparable with those achieved by more sophisticated approaches such as neural networks and identification trees [1, 5]. This accuracy is highly dependent on the function used to measure the similarity between instances. To measure the similarity between instances, a similarity (distance) function is used [5 (see also [8] for an excellent survey of such functions). The function that we use in this work is defined below. It is a variant of the HVDM

$$HVDM(X,Y) = \sqrt{\sum_{a=1}^{m} d_a^2(x_a, y_a)}$$

where X and Y are two instances, m is the number of attributes, x_a and y_a are the values for attribute a in instances X and Y respectively. The distance function d_a between two values depends on the type of the attribute symbolic or numeric. It is defined as follows

$$d_a(x,y)\begin{cases} vdm_a(x,y), & \text{if } a \text{ is symbolic else} \\ \dfrac{|x-y|}{a_{\max} - a_{\min}} & \text{if } a \text{ is numeric} \end{cases}$$

[1] KASIT, University of Jordan, Amman, Jordan. Email: hindi@ju.edu.jo

$$vdm_a(x,y) = \sum_{a=1}^{C} \left(\frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}} \right)^2$$

where:

- $a_{\max}$ and $a_{\min}$ are the maximum and minimum values of attribute a
- $N_{a,x}$ is the number of instances in the training set that have the value x for the attribute a.
- $N_{a,x,c}$ is the number of instances in the training set that have value x for attribute a and belong to output class c.
- C is the total number of output classes in the problem domain

III. Early Stopping

In [2] we presented some early stopping criteria for the knn algorithm that allow it to find reasonably close neighbors (but not necessarily the nearest neighbors) for classification purposes without having to search the whole instance space. A simple early stopping criterion finds, for each instance in the instance memory (training set), the distances to its nearest enemy, which we call Nearest Enemy-Instance Distance NEID. When classifying a new instance, the knn algorithm halts once it finds k close instances. An instance is considered close enough to the new instance if the distance between it and the new instance is smaller than the distance between it and its nearest enemy (i.e., less than its NEID).

IV. Interval Indexing Trees

In traditional databases, indexing refers to the task of mapping the record key to the storage location [3]. Several techniques have been used for this purpose such as hashing, B trees, and R trees. However, there is an important difference between traditional indexing and indexing in instance based learning [3]. Traditional indexing is based on exact matching i.e., a given value is either present or not present. While indexing instances, should be content-aware or approximate in that if it cannot find an exact matching value it should return the location of the nearest value.

In this section, we describe how interval trees can be used for this approximate matching. The idea is to estimate the distance between two instances i and j relative to a base instance *b*, using the formula

dist(i,j)≈dist(b,i)-dist(b,j)

Of course, this only gives an approximate estimation of the distance between i and j. However when classifying a new instance, it gives a clue about the instances that are more likely to be close to it and should, therefore, be considered first.

To determine these possibly close instances, an interval tree such as the one shown in fig 1, is used. The tree indexes all instances in the training set according to their distance from the base instance.

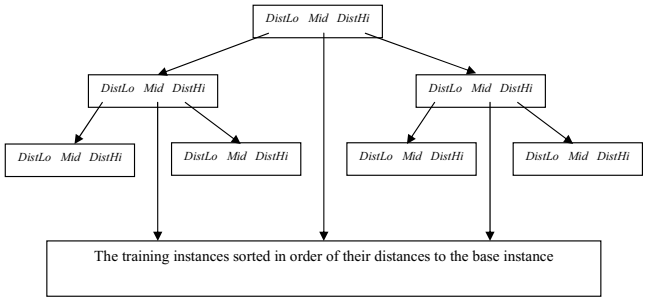

Fig 1 the general structure of an interval tree.

Each node in the tree represents a cluster of nearby instances that belong to the same class. It can also be seen as representing a distance range. A node stores the distance between the base instance and the nearest and farthest instances in the clusters, *DistLo* and *DistHi* respectively. It also stores the location of the instance at the center of the cluster, *Mid*.

```
function Build_interval_tree(instances,m,b)
// instances is the training set
// m is the number of instances in the training set
// b is the base instance
begin
  compute the distance between each instance in instances
and the base instance, b.
  sort the instances according to their distance to the base
instance
  return TreeBuild(1,m)
end

function TreeBuild(lo,hi)
// lo and hi are the left and right boundaries of the instance
region for which the tree is
//being build
begin
  if lo<=hi
  determine the median cluster, in the region between lo and
hi
  let mid be the location of the center instance of the median
cluster.
  left=the location of the leftmost element in the cluster;
  right=the location of the rightmost element in the cluster;
       root.DistLo=distance(left,Base);
       root.DistHi=distance(right,Base);
       root.mid=mid;
      // build the left subtree
      TreeBuild(lo,left-1);
      // build the right subtree
      TreeBuild(right+1,hi);
  end if
end
```

Fig 2 The interval tree construction algorithm

Table I shows the classification accuracy (Acc) and the proportion of training set searched (Size).

	knn		interval trees			
	k=1	k=3	k=1		k=3	
Training set	Acc%	Acc%	Acc%	Size%	Acc%	Size%
Zoo	96.7	94.4	91.1	3.8	93.3	10.0
Wine	95.5	96.1	90.5	5.8	94.4	16.0
Iris	96.0	95.3	95.3	3.0	95.3	7.4
Flag	70.7	70.7	68.8	22.8	70.7	53.3
Sonar	85.5	82.2	82.7	17.4	86.6	39.7
Lymph	50.0	52.1	51.3	37.2	54.1	70.8
Vehicle	68.8	66.4	66.6	11.9	69.0	44.1
Hepatitis	80.5	80.0	79.2	14.8	81.1	34.5
horse-colic	72.4	72.8	74.4	27.9	74.1	57.5
pima-indians-diabetes	70.8	72.0	73.1	15.1	72.7	45.8
Promoters	92.5	94.3	91.4	6.1	93.4	15.5
liver_bupa	62.9	61.4	59.7	19.9	63.5	66.1
heart-hungarian-2	74.5	77.9	78.1	16.9	79.5	38.9
heart-long-beach-va-2	64.0	73.0	67.0	19.3	68.5	52.9
Heart	77.0	83.7	79.3	13.6	81.5	42.1
heart-cleveland-2	75.6	80.9	76.9	16.5	79.2	39.4
Average	**77.1**	**78.3**	**76.6**	**15.7**	**78.6**	**39.6**

To construct an interval tree we begin by sorting the instances in ascending order according to their distances from the base case. This step gives a list of clusters; each cluster consists of nearby instances that belong to the same class. Each cluster is represented by a tree node. To ensure that the resulting tree is height balanced, the cluster that is represented by the root of the tree is the median cluster. The algorithm calls itself recursively to construct left and right subtrees, representing the clusters to the left and right of the cluster represented by the root. The details of the algorithm are shown in fig 2. The resulting tree is a balanced tree which makes the tree search time of order $\log_2 n$, where n is the number of clusters.

To classify a new instance, its distance from the base case, d_b, is measured. Next, the interval tree is used to determine the range in which the distance falls and the location of the center instance, *Mid*, of the corresponding cluster. If the distance, d_b, lies in no range the closest range is returned.

The search for the nearest instance starts at location *Mid* and spreads in both directions until the early stopping criterion is met (i.e., until an instance, that finds the new instance closer to it than its enemy, is found). In the worst case, the whole instance space is searched for the nearest instance. However, as our empirical study shows this not the usual case.

Choosing a suitable base instance is important. A good base instance should give a good approximation of the distance between a pair of instances i and j. We use an artificial (not necessarily in the training set) base instance, that is constructed from the most common values for discrete attributes and the median value for continuous attributes.

I. Results

To study the effectiveness of the discussed method, an empirical work using 16 data sets, obtained from the Machine Learning Repository at the University of California Irvine, was conducted. Ten-fold cross validation was used in all experiments. Table-I shows a summary of the results. The first column shows the name of the training set, the second and third columns show the classification accuracy obtained using the knn algorithm (searching all the training sets) with k=1 and k=3. The remaining columns show the classification accuracy and the searched proportion of the training set, using interval trees with k=1 and k=3.

The table shows that with k=1, and using interval trees we achieved an average classification accuracy of 76.6% which is very close to the classification accuracy obtained using the original knn algorithm (also with k=1) which is 77.1%. However using interval trees and the discussed early stopping criterion the algorithm had to search on average only 15.7% of the training sets. However, using interval trees with k=3 we had to search a larger proportion

of the training set. On average, it searched 39.6% of the training set, achieving a slight increase in the average classification accuracy of 78.6% compared to that achieved using the knn algorithm with k=3, which is 78.3%.

II. Conclusion

This work presented an approximate indexing method that uses simple interval trees to determine a good starting point to search the instance space for reasonably close instance(s). The search terminates once an early stopping criterion is met. Our empirical study using 16 different training sets show a great reduction in the proportion of the training set that needs to be searched while maintaining comparable classification accuracy to that obtained by the knn algorithm searching the whole training sets. Future work may investigate the use of other early stopping criteria and other methods to determine the base instance

References

[1] S. Cost, and S. Salzberg, (1993) A weighted Nearest Neighbor Algorithm for Learning with Symbolic Features. Machine Learning, Vol. 10, pp. 57-78.
[2] Hindi, K (2003) Early-Halting Criteria for Instance-Based Learning, in Proc of ACS/IEEE International Conference on Computer Systems and Applications, AICCSA03, Tunisia.
[3] S. K. Pal, S.C. K. Shiu, Foundations of Soft Case-Based Reasoning, John Wiley & sons.2004.
[4] Sproull, Robert F. (1991). Refinement to Nearest-Neighbor Searching in K-Dimensinal Trees. Algorithmica, Vol. 6, pp. 579-589.
[5] C. Stanfill, D. Waltz, (1986). Toward memory-based reasoning. Communications of ACM, 29 No 12, pp 1213-1228.
[6] Wess, Stefan, Klaus-Dieter Althoff and Guido Derwand, (1994). Using k-d Trees to Improve the Retrieval Step in Case-Based Reasoning. Stefan Wess, Klaus-Dieter Althoff, & M. M. Rithcher (Eds.), Topics in Case-Based Reasoning. Berlin: Springer-Verlag, pp. 167-181.
[7] D. R. Wilson, T. R. Martinez (2000). Reduction Techniques for Examplar-Based Learning Algorithms. Machine Learning 38, pp. 257-268.
[8] D.R. Wilson , T. R. Martinez (1997). Improved Heterogeneous Distance Functions. Journal of AI Research, 6, pp. 1-34.